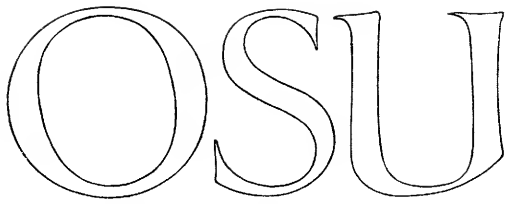
cc-69-15

OSCAR V55: With Character String Processing

by
Gilbert A. Bachelor

June, 1969



COMPUTER CENTER
Oregon State University
Corvallis, Oregon 97331

OSCAR V55:
With Character String Processing

by

Gilbert A. Bachelor

cc-69-15

Computer Center
Oregon State University
Corvallis, Oregon 97331

June, 1969

OSCAR V55: With Character String Processing

A new version of OSCAR (V55) has been released. The most important new feature of V55 is the improved character string processing facility. A full discussion of this facility appears later in this report. First, we shall describe the other changes and improvements in OSCAR V55.

CHANGES, IMPROVEMENTS AND CORRECTION OF BUGS

1) There were two bugs involving procedures in version 54. One of them was that OSCAR failed to detect an attempt to define a procedure with more than 13 parameters. It did not give an error message; it simply gave an incorrect result. V55 detects this situation and prints an error message. The other bug was that if F (for example) were defined as a procedure with one or more parameters, then one could not re-define F as a procedure without parameters, without first CLEARing F. This bug has been fixed.

2) Considerable effort has been expended to shorten OSCAR by re-coding numerous portions of it. This effort has succeeded well enough so that V55 is shorter (takes less memory space) than V54, in spite of the new features that V55 has. One of the changes made in this "shortening" effort has been to change and shorten many of the error messages. For example, there were three different messages concerned with lines that were too long: INPUT LINE TOO LONG; INPUT RECORD TOO LONG; and LINE TOO LONG. All three of these situations now produce the message LINE TOO LONG.

3) The READCHAR statement now reads an empty character string only at the end of a line. (For more information about READCHAR, see the discussion on character string processing, later in this report.)

4) Protected output unit: In version 54, if one used a command &OUTPUT, (lun) and the specified logical unit were pro-

tected, OSCAR would simply switch to the teletype for output instead of trying to write on the unit. This action tended to cause confusion. In OSCAR V55, if the output unit is protected, OSCAR will go ahead and try to write on it, which will produce an OS-3 error message. This situation will normally not occur, however, since the &OUTPUT,(lun) command now checks for protection on the specified output unit and produces an error message if the unit is protected.

5) The KIND function produces a character string telling what "kind" of object its argument is. For example, KIND(4) is "INTEGER". The spelling of some of the "kinds" has been changed in version V55. For example, the spelling of RATIONL has been changed to RATIONAL.

6) The square root function (SQRT) has been re-coded to make it shorter, faster, and more accurate. One result of this is that SQRT in V55 will produce slightly different answers, in some cases, than V54's SQRT. Also, since SQRT is used in LOG (when the argument is a complex number), and in ABS (when the argument is an array or a complex number), these functions will also give different answers in some cases.

7) When OSCAR V55 is reading data input (&INPUT,lun) from a file and it comes to end of file or end of data, it prints a message of the form (lun) AT EOF, and behaves as if interrupt (MI) had occurred. In V54, OSCAR printed a message and switched data input to the teletype, which was usually more confusing than useful.

8) If OSCAR V55 is attempting to read a very long line and cannot because of a storage overflow situation, it prints the message LINE TOO LONG and behaves as if interrupt (MI) had occurred. OSCAR V54 printed the message and waited for a new line from the teletype to replace the long line. Again, this tended to be confusing.

9) When OSCAR is reading from a file (either CONTROL or INPUT), there may be several blanks (spaces) at the end of each

record. This caused a problem when one was doing a READLINE since one would not know how many blanks there might be. To alleviate this problem, OSCAR V55 eliminates all trailing blanks from records read from files except in the case where the record is all blank. In this case, the record will contain one blank.

10) As an aid to CRT users, the notation :== is now allowed as a substitute for ←=. Both of these notations represent the "old value assignment". The value of an old value assignment is the old (previous) value of the variable on the left. For example, (X:==(X+Y)/2)EQ Y will assign the new value (X+Y)/2 to X and then compare the previous value of X with the value of Y to see if they are equal. Another example: F1:=F2:=1; FOR I=1 TO 100 PRINT F1:=F2:==F1+F2,CR will print a table of Fibonacci numbers. (F1+F2 is assigned to F2, the previous value of F2 is assigned to F1, and this value is also printed. If the PRINT statement were PRINT F1:==F2:==F1+F2,CR then the previous value of F1 would be printed.)

11) The "null" feature introduced in V54 has been more fully implemented. A sub-array designator such as A(5:3) (first number greater than second one) produces a "null" result, which when added, subtracted, multiplied, etc., by anything else, is supposed to produce the other operand as a result. In V55, there are still some cases where this does not work, but it works in more situations than it did in V54.

12) Assignment operators (:=, :==, ::=) can now be used on either side of a colon (:) operator, and they have a higher precedence than colon. For example, A(I:=I+1:J:=J-1) is now legal. (This increments I, decrements J, and uses the new values to select a sub-array of A.)

13) The "storage management" in OSCAR V55 has been changed in several ways to try to alleviate the "fragmentation" problem. OSCAR uses storage in blocks of words of various sizes, usually powers of two (2, 4, 8, 16, etc.). A block which has been broken

into smaller blocks cannot be put back together again. As a result, there sometimes occurs a situation where a large block of storage is needed and there are none available, even though there might be enough storage scattered around in smaller blocks. In this situation OSCAR proclaims a STORAGE OVERFLOW and asks the user if it is okay to get another page (2048 words) of storage. We have made a number of changes in V55 to reduce the amount of storage fragmentation. There are further changes that should be made and work will continue on this problem. Users who have large OSCAR programs that run into storage overflow problems can alleviate the situation by having the program erase parts or steps that are no longer needed and by CLEARing arrays that are no longer needed. For example, one could put a statement 12.99:ERASE PART 12 at the end of part 12. One should also try to be sure that the program is not "recursing" unintentionally. Use a GO TO where possible, in preference to DO PART or DO STEP, since a DO PART or DO STEP sends everything down to a "lower level", consuming more storage.

14) Two minor changes have been made in the OSCAR display console routines. One of these allows the CRT user to clear the screen by pressing CLEAR, then SEND. (Previously, OSCAR would clear the screen if the user merely pressed SEND but this made it too easy to clear the screen inadvertently.) Another change is that if a blotch (■) is read in (not in upper left corner), it is interpreted by OSCAR as a quote mark. This makes OSCAR'S CRT input and output compatible for those users who like to read statements displayed on the screen back into OSCAR.

15) Previous versions of OSCAR allowed zero divided by zero (0/0) to slip through, giving an answer of 0 instead of an error message. In OSCAR V55, this case will now produce an error message. However, integer zero (0) divided by an inexact zero (OE-5P1, for example) still gives an answer of 0. This should be fixed in the next version of OSCAR.

NEW FEATURES

1) Comments are now allowed in stored program steps. For example, 1.05:*COMPUTE THE SUM is now legal. Executing such a step does nothing; the value of the step is [] (undefined). (In version 55, stored program steps with comments have to be typed in individually; they can't be typed in when using &PROGRAM unless one types a space before the *).

2) OSCAR can now be used in a batch job. If OSCAR is called by the statement 7OSCAR, it will treat unit 60 (the card reader) as its "teletype" for input, and unit 61 (the line printer) as its "teletype" output unit. All statements, commands and data read from the input unit will be printed on the output unit, with a mode indicator at left. Outputs from OSCAR to its "teletype" will be printed on the output unit, with no mode indicator. A record read from the input unit that contains only the word ESCAPE, starting in column 1, will cause OSCAR to behave just as it does when a teletype user presses the ESCAPE key (at the beginning of a line). (ESCAPE changes modes or interrupts data input).

If OSCAR is called by a statement of the form 7OSCAR,I=(lun), then OSCAR will use the specified logical unit number as its "teletype" input unit and will otherwise behave as described above. The input unit must already be equipped before OSCAR is called.

A file mark (or end of file card) on the input unit denotes the end of input for OSCAR, and it will print END OF OSCAR RUN and return to control mode. If any error occurs and the user is not using DEBUG (see next section), OSCAR will abort its run with the message **ABNORMAL END OF OSCAR RUN. (This is printed after the error messages.)

3) The DEBUG command is not new but it has a new feature for the convenience of batch users of OSCAR. If a batch user of OSCAR does not want the run aborted in case of an error, he should include the command &DEBUG ON or the command &DEBUG ON (lun).

(The & is printed as a ≤ on the line printer.) If the command DEBUG ON has been used, then a subsequent error will produce a special message of the form ERROR AT (addr) (message), a few blank lines, then the normal error messages and OSCAR will go on reading from its input unit instead of quitting. If the command DEBUG ON (lun) has been used, then an error will produce the special message mentioned above. OSCAR will then read commands (commands only) from the logical unit specified in the DEBUG ON statement. Each command read will be printed and then executed. This will continue until the command GO is read or a record is read containing only the word ESCAPE. OSCAR will then print the regular error messages and go on reading from its input unit.

Teletype and CRT users can also use the DEBUG feature, with the additional provision the OSCAR will read commands from the console if an error occurs after a DEBUG ON command has been given. An "escape" will terminate this special command mode. The DEBUG ON (lun) works just as described above.

To cancel the debug mode, use the command DEBUG OFF.

4) When OSCAR is reading from a file (or a card reader, in a batch job) it will recognize a line containing only the word ESCAPE. If OSCAR is reading statements or commands from the file, ESCAPE causes it to switch from normal to command mode or vice versa. If OSCAR is reading data from the file, ESCAPE causes an "interrupt" just as if MI had occurred. Whenever the &CONTROL, (lun) command is given OSCAR will start reading from the (lun) in normal mode. At the end of a file, OSCAR switches back to the standard input unit (console, card reader, etc.) and also reverts to normal mode.

5) For CRT users, there is now a RECORD feature, which enables the user to get a "hard copy" of everything he does in OSCAR. In conjunction with this is a "recall" feature, which enables the CRT user to bring back to the screen a display of what he has done earlier.

The RECORD feature can be activated in either of two ways. The easiest way is to call OSCAR with a statement of the form ≠OSCAR,R=(lun). This will cause OSCAR to record on the specified (lun) and this (lun) will be equipped as a file if it is not already equipped. The other way to start recording is to use the command <RECORD ON (lun). This will cause OSCAR to record on the specified (lun), which must have been equipped previously. The record feature can be turned off by the command <RECORD OFF. While the Record feature is on, OSCAR displays an "R" next to the second mode indicator at the right side of the screen.

When "Record" is on, every line displayed on the screen, whether it is an input by the user or an output from OSCAR, will be written on the specified record unit. The first two characters of each recorded line are blanks and the lines are from one to thirteen words long (4 to 52 characters). (Trailing blanks are removed.) This output can thus be written directly on a line printer or can be written on a file and later copied to a line printer, if desired.

The "recall" feature mentioned above is available only when the Record feature is in use. Also, the record unit must be a file (since one cannot backspace a line printer). If one is recording on a file and if one is in normal mode ("blotch" displayed in corner), then one can "recall" anything which has been recorded on the file. To do this, one types a special "command" whose first character is the "blotch". The SKIP key is used to space past the blotch without wiping it out. Then one types either a decimal integer or a minus sign followed by a decimal integer and presses SEND. If an unsigned integer was typed, OSCAR will rewind the Record unit then space forward as many records as specified by the integer. It will then read the next sixteen records from the file and show them on the screen. Finally, it searches forward to the end of data on the file, to be ready for further recording, etc.

If a negative integer was typed, OSCAR backspaces the file the number of records specified by the integer. It then acts as described above (reads forward sixteen records, shows them on the

screen, etc.).

If, for any reason, the recall command cannot be carried out, OSCAR does nothing. If the file is positioned to a point less than sixteen records from its end, then fewer than sixteen lines will be shown on the screen. (Note: the recalled information is not recorded on the file again.)

One convenient use of the recall feature is to position a previously typed statement near the top of the screen, carriage return down to it and read it back into OSCAR, possibly after making changes in it.

CHARACTER STRING PROCESSING

Previous versions of OSCAR had some ability to process character strings (add, subtract, compare). Version 55 has several new features which make it possible to do general character string manipulations. These features include subscripting and sub-array notation to refer to parts of character strings, a search feature and DECODE and ENCODE functions. To indicate the possibilities, three simple applications which have been programmed for OSCAR V55 are a "form letter" processor, a Markov Normal Algorithm processor and a simple expression translator (from algebraic expressions to Polish strings). We shall describe OSCAR's character string processing features (both old and new) on the following pages.

INPUT, OUTPUT AND REPRESENTATION OF CHARACTER STRINGS

The internal representation of character strings in OSCAR has been changed. In previous versions, a string was terminated by a quote mark ("). This meant that a quote mark could not be included in a string. The new representation includes an integer (internally) that tells how many characters the string contains. This makes it possible to include any characters in a string.

There are two ways of representing a character string in OSCAR language. One way is to enclose it in quotation marks, as

for example: "THIS IS A STRING". This notation is fine for teletype users. There is no quotation mark on the CRT keyboard; however, OSCAR, when used at a CRT, allows the notation ≠≠ to represent a quotation mark. (On output to the CRT, a quotation mark is displayed as a blotch.) Unfortunately, EDIT does not recognize the ≠≠ notation and treats it as two apostrophes in a row (''). A CRT user who wants to use EDIT to prepare or to modify an OSCAR program cannot use the quotation mark at all. To alleviate this problem, there is another notation for character strings: TEXT /THIS IS A STRING/ means the same as the previous example. In this second notation, the word TEXT can be followed by any non-space character (for example: / + : $, etc.), then the string of characters and finally, the chosen "bracket" character again. Thus, TEXT/X=/, TEXT:X=: and TEXT%X=% all represent the same character string "X=".

There are three special conventions regarding the contents of character strings. These are: 1) The bracketing character (" or whatever bracketing character is used with TEXT) can be included within a character string by writing it twice in a row. For example, "X""Y" represents the character string X"Y. Other examples; TEXT =X=== represents the string X=, """" represents the string ", and "" represents the empty string. 2) Carriage return (end of line) <u>always</u> terminates a character string and the carriage return is included in the string as its last character. Thus, if PRINT "RESULTS ARE: is the last thing on a line, the character string to be printed will be RESULTS ARE:, followed by a carriage return. 3) The character @ (↓ for CRT users) denotes a carriage return <u>within</u> a character string. For example:

"FIRST LINE@2ND LINE@LASTLINE@"

The three @ symbols in this character string represent carriage returns. As a result of this convention, it is not possible to type in a character string that actually has the character @ (or ↓) in it. The only way to get a character string with @ in it is to use the ENCODE function (see later paragraph).

When a character string is printed (using PRINT), the characters in the string are simply typed out, with no bracketing characters, and carriage returns cause output to go to the next line. If EPRINT is used, the bracketing characters " " are printed; quotation marks within the string are printed twice, and carriage returns are printed as @. The result is that an EPRINTed character string can be read back in and will produce a character string identical to the one that was EPRINTed (unless it contained a @ character; such a character will go back in as a carriage return). The constant CR is a special case; this represents a one-character string containing a carriage return but it will be printed as a carriage return whether PRINT or EPRINT is used.

There are three ways in which character strings can be read in by OSCAR.

1) A statement such as READ X will read the next constant from the input unit and assign it to X. The constant can be a number, array, etc., or a character string, of either the "..." form or the TEXT/.../ form.

2) The statement READLINE X will read an entire line (or the rest of a partially read line), put it into a character string and assign it to X. In this case, the carriage return at the end of line will not be included in the character string.

3) The statement READCHAR X will read the next single character from the input unit and assign it to X as a one-character string. At the end of a **line**, READCHAR will get an empty character string instead of a string containing a carriage return.

The statements READ, READLINE and READCHAR may each include a list of variables to be read. For example, READLINE X, Y, Z will read three lines and assign them to the variables X, Y and Z.

OPERATIONS ON CHARACTER STRINGS

Character strings can be "added", "subtracted" or "compared".

X+Y If X and Y are character strings, X+Y is a character string composed of X followed by Y (concatenation). For example, "STREET" +"CAR" is the string "STREETCAR".

X-Y If X and Y are character strings, X-Y is found by searching X to see if Y occurs within it (as a substring). If so, the part of X that matches Y is removed, and the value of X-Y is what is left. (X and Y are not changed.) If Y is not found in X, the value of X-Y is simply X. Another way of expressing X-Y is SAR(X,Y,""), that is, search and replace by empty (see next section for SAR function). For example, "CHARACTER"-"AC" is "CHARTER". Another example: "WALLA WALLA"-"ALL" is "WA WALLA" (only the first occurrence of "ALL" is removed).

X EQ Y If X and Y are character strings, X and Y are compared character by character to see if they are the same. If X and Y are the same length, and all corresponding characters match, then the value of X EQ Y is TRUE; otherwise, the value is FALSE.

X NEQ Y This is like X EQ Y except that it gives the result TRUE if X is not the same as Y; if X and Y are the same, the value is FALSE.

X<Y If X and Y are character strings, then corresponding characters of X and Y are compared, starting at the left, until two corresponding characters are different or the end of one (or both) of the strings is reached. If the strings are of the same length and all characters of X match the corresponding characters of Y, the value of X<Y is FALSE. If X is shorter than Y, but X matches Y as far as it goes, the value of X<Y is TRUE. If Y is shorter than X, and matches X as far as it goes, then X<Y is FALSE. If there is some position where a character in X does not match the corresponding character of Y, then X<Y is TRUE if the internal code for the character in X is less than the code for the character in Y; otherwise, X<Y is FALSE. (See table of codes at the end of this report.)

X GEQ Y This is the same as X<Y except that it gives the answer TRUE if X<Y is FALSE, and vice versa.

X>Y This is the same as Y<X.

X LEQ Y This is the same as Y GEQ X.

FUNCTIONS WHICH CAN BE USED WITH CHARACTER STRINGS

Of course, the OSCAR user can define functions of his own which take character strings as arguments or produce character strings as values, or both. However, there are several "pre-defined" functions available in OSCAR which can be used with character strings. These are:

DECODE If X is a character string, then DECODE(X) is an integer in the range 0 to 63, giving the internal (OSCAR) code for the first character of the string. If X is an empty character string, then DECODE(X) has the value -1.

ENCODE If N is an integer in the range 0 to 63, then ENCODE(N) is a one-character string containing the character whose internal code is the specified integer.

SRCH If X and Y are character strings, then SRCH(X,Y) is an integer telling where string Y occurs within X, if it does occur. If it does not occur, the value of SRCH is zero. For example, SRCH ("ABCDE","CD") has the value 3. (CD occurs within ABCDE, starting at the third character of ABCDE.) If Y occurs in X in more than one place, SRCH indicates the location of the left-most occurrence of Y.

SAR If X, Y and Z are character strings, the value of SAR(X,Y,Z) is a character string formed in the following manner: X is searched for an occurrence of Y (using SRCH); if found, a new string is constructed in which the part of X that matched Y is replaced by Z. This new string is the value of SAR. If Y does not occur within X, the value of SAR is simply X. In any case, X, Y and Z are not changed. For example, SAR("OSCAR","CAR","WALD") has the value "OSWALD".

SIZE If X is a character string, SIZE(X) is an integer (0 or larger) telling how many characters the string X contains.

KIND KIND(X) produces a character string telling what kind of value X has.

Note: The arguments of the functions described above can be any expressions, so long as the expressions have the proper kinds of values.

SUBSCRIPTING AND SUB-ARRAYS

Subscript notation can be used to refer to individual characters in a string and the sub-array notation can be used to refer to sub-strings of a string.

X(I) If X is a character string and I is an integer in the range $1 \leq I \leq$ SIZE(X), then X(I) is a one-character string containing just the Ith character of X. If I = 0 or if I > SIZE(X), then X(I) is an empty character string. For example, if X = "ABCDE", then X(4) is "D" and X(6) is "".

X(I:J) If X is a character string and I and J are integers such that $1 \leq I \leq J \leq$ SIZE(X), then X(I:J) is a character string containing the Ith through Jth characters of X, inclusive. If any of the three inequalities specified above is not true, then X(I:J) is an empty character string. For example, if X = "ABCDE", then X(2:4) is "BCD", X(4:6) is "", and X(4:1) is "".

Note: One cannot assign values to x(I) or to X(I:J), when X is a character string. For example, X(3):="Q" is illegal if X is a character string. One can accomplish the effect desired in this example by the statement

```
X:= X(1:2) + "Q" + X(4: SIZE(X))
```

OTHER NOTES ABOUT USE OF CHARACTER STRINGS

As we have implied in the foregoing paragraphs, variables may have character strings as their values. This can happen either by reading in a character string or by an assignment statement (such as the example above). One can use either ordinary assignment (:=) or the old value assignment (:==) in dealing with character strings. One can also use the exchange operator (==).

Another useful concept to keep in mind, is that the elements of an array can be character strings. For example, A(1):="THIS"; A(2):="IS"; A(3):="AN"; A(4):="EXAMPLE." If the array A has only these four elements, then PRINT A will produce the sentence THIS IS AN EXAMPLE. Also, A(4,1:4) is the string "EXAM". (This selects the first through fourth characters of the fourth element of A.) Another way to define an array containing character strings is to use an ARRAY constant. For example, B:=ARRAY("THIS", "IS", TEXT/ANOTHER/,"EXAMPLE.") (One can use either the "..." or the TEXT notation in an ARRAY constant.) If B is PRINTed, the result is THIS IS ANOTHER EXAMPLE. And finally, PRINT B-A would produce OTHER. (Actually, B-A is an array whose Ith element is B(I)-A(I), for I=1,2,3,4. But, three of these elements are empty character strings; only the third one actually contains any characters.)

OSCAR internal character codes

The following table gives the character codes used in OSCAR. These codes determine the values of the DECODE and ENCODE functions, and also determine the results of comparisons such as <, GEQ, etc. (TTY denotes the teletype character; CRT denotes the display console character; and LP denotes the Line Printer character.)

OSCAR code	card code	TTY char	CRT char	LP char	OSCAR code	card code	TTY char	CRT char	LP char
0	0	0	0	0	32	0,6	W	W	W
1	1	1	1	1	33	0,7	X	X	X
2	2	2	2	2	34	0,8	Y	Y	Y
3	3	3	3	3	35	0,9	Z	Z	Z
4	4	4	4	4	36	12,3,8	.	.	.
5	5	5	5	5	37	11,6,8	@	↓	↓
6	6	6	6	6	38	5,8	&	≤	≤
7	7	7	7	7	39	11,0	!	∨	∨
8	8	8	8	8	40	(none)	carriage return		
9	9	9	9	9	41	0,7,8	?	∧	∧
10	12,1	A	A	A	42	12,5,8	#	≥	≥
11	12,2	B	B	B	43	4,8	'	≠	≠
12	12,3	C	C	C	44	12,6,8	"	■	¬
13	12,4	D	D	D	45	12,0	<	<	<
14	12,5	E	E	E	46	3,8	=	=	=
15	12,6	F	F	F	47	11,7,8	>	>	>
16	12,7	G	G	G	48	12	+	+	+
17	12,8	H	H	H	49	11	-	-	-
18	12,9	I	I	I	50	11,4,8	*	*	*
19	11,1	J	J	J	51	0,1	/	/	/
20	11,2	K	K	K	52	11,5,8	↑	↑	↑
21	11,3	L	L	L	53	0,6,8	←	'	≡
22	11,4	M	M	M	54	0,4,8	(	(	(
23	11,5	N	N	N	55	12,4,8	)	)	)
24	11,6	O	O	O	56	7,8	[	[	[
25	11,7	P	P	P	57	0,2,8	]	]	]
26	11,8	Q	Q	Q	58	2,8	:	:	:
27	11,9	R	R	R	59	0,3,8	,	,	,
28	0,2	S	S	S	60	12,7,8	;	▲	;
29	0,3	T	T	T	61	blank	space		
30	0,4	U	U	U	62	11,3,8	$	$	$
31	0,5	V	V	V	63	6,8	%	%	%